instantiated by a createInstance method of BusinessService class 48. The createInstance method preferably uses a Java Reflection application programming interface (API) to find a class with the same request name as the value of the request name parameter. The Java Reflection API supports dynamic retrieval of information about classes and data structures by name, and allows for their manipulation within an executing Java program. The createInstance method instantiates the class and passes a reference to the instance back as a subclass of BusinessService class 48. The reference is in the form of a name that is the same as the request name.

Translation Of Request Parameters To An Input Message

Using the instance of the subclass of the BusinessService class 48, the doGet method calls a createInputMessage method of the BusinessService class 48 at block 108. At block 110 the createInputMessage method creates an input message in XML format that is compatible with the instance of the subclass of the BusinessService class 48. In one embodiment, the createInputMessage method creates an instance in a subclass of Message class 44, and labels the input message with a message name that is the name of the instance of the subclass of BusinessService class 48 plus the suffix "_REQUEST".

Referring now to FIG. 4, at block 112, a constructor for Message class 44 creates a first DOM document within an instance of the DOM Document class. The first DOM document represents the input message. The constructor for Message class 44 sets a root element node of the first DOM document to be a new Field (DOM Element) at block 114. At block 116 a createField method of Message class 44 is executed in ApiService class 42. The createField method creates the input message by adding element nodes and corresponding text nodes in the first DOM document. The element and textnodes are created as a function of the tags present in the request. The createField method also sets the text nodes to the unit of data associated with each of the tags in the request. The createField method is preferably provided in a plurality of different versions in Message class 44 as detailed in the computer program listing appendix filed herewith. The plurality of different versions may support different datatypes such as, for example, the previously described short integer, long integer, Boolean, string and group datatypes.

In one embodiment, one version of the createField method is used to create fields for the input message from the request. Specifically, only the createField (String, String) version is used to create element nodes for each tag and set each of the text nodes to the corresponding unit of data. One version of the createField method is used in this embodiment since the text node of each element is set to all the text (unit of data) identified by a tag in the request regardless of datatype. Accordingly, this embodiment does not make use of the datatypes included in MESSAGEDEFINITION class 50. In other embodiments, the datatype may be used to execute different versions of the createField method as a function of the datatype. A plurality of versions of the createField method may be used during processing to create the output message as discussed in detail later.

At block 118 the createField method creates an instance of Field class 46. The instance of the Field class 46 wraps an instance of DOM Element class. Wrapping DOM Element class involves the constructor for Field class 46 storing a reference to the new field (DOM Element) passed to Field class 46 in an instance variable. The createField method sets a tag "message" as the field name for the root element node forming the top level in the DOM virtual tree hierarchy at block 120.

At block 122, a setAttribute method in Field class 46 is executed. The setAttribute method is a wrapper for the DOM setAttribute method of DOM Element class. Similar to the previously discussed createField method, the setAttribute method is provided in a plurality of different versions in Field class 46 as detailed in the computer program listing appendix filed herewith. The plurality of different versions may support different datatypes such as, for example, the previously described short integer, long integer, Boolean, string and group datatypes. In one embodiment, one version of the setAttribute method is used to set attributes for the root element of the input message. In this embodiment, only the setAttribute (String, String) version is used since the attribute node of the root element is set to all the attributes within the tag regardless of datatype. It should be noted by the reader that the remaining elements (tags) in the request of this embodiment do not include attribute nodes. In other embodiments, however, the request may include elements with attribute nodes. In addition, the reader should note that a plurality of versions of the setAttribute

method may be used during processing to create the output message as discussed in detail later.

Referring now to FIG. 5, the attribute node of the root element node is set by a version of the setAttribute method in Field class 46 at block 124. More specifically, an attribute name of the attribute node is set to "name" and an attribute value is set to the message name of the input message (e.g. (message name)_REQUEST). Setting the root element node to the message name establishes the basic structure of the input message as an XML structure (one element, with all the remaining fields contained as sub elements). Setting the root element node also ensures that messages are compliant with the XML standard of having a single root element.

The string literals (datatype = string) used in both the createField method and the setAttribute method are preferably loaded from a static declaration of the datatype contained in MESSAGEDEFINITION class 50. The static declaration of the datatype identifies the format used for translation of the request parameters to the input message.

At block 126 a pickName method of Message class 44 is executed. The pickName method operates to pick the version of the field name as a function of the mode debug flag. The selected version of the field name is utilized when adding tag names to the input message as previously discussed.

The doGet method of ApiService class 42 processes the request parameters passed in the request and uses the createField method to create additional fields (element nodes and text nodes) at block 128. Because the root element node of the input message was previously created, these invocations of the createField method append the new fields as children to the "message" field name.

Execution Of Subclasses of BusinessService class 48

At block 130, the translation of request parameters to form the input message is complete and the doGet method of ApiService class 42 calls a Launch method of BusinessService class 48. The Launch method executes a createOutputMessage method of BusinessService class 48 at block 132.

Referring now to FIG. 6, at block 136, the createOutputMessage method creates an output message in an instance of a subclass within Message class 44 similar